CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

Evaluating Stormwater Flow in an Urban Environment using GIS

A Case Study in Glendale, California

A graduate project submitted in partial fulfillment of the requirements

For the Master of Arts in Geography (GIS)

By

Chad March

May 2014

The graduate project of Chad March is approved:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Dr. Reagan Mass Date

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Dr. Steve Graves Date

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Dr. Soheil Boroushaki, Chair Date

California State University, Northridge

Table of Contents

[Section 1: Introduction 1](#_Toc378521283)

[1.1 Land-use influences 3](#_Toc378521284)

[1.2 Natural Pollutants 3](#_Toc378521285)

[1.3 Pollutant Origins 4](#_Toc378521286)

[1.4 Monitoring Water Quality 5](#_Toc378521287)

[1.5 Watershed Management 6](#_Toc378521288)

[1.6 Study Area 7](#_Toc378521289)

[Section 2: Background 10](#_Toc378521290)

[2.1 Current Models of Stormwater Analysis 10](#_Toc378521291)

[2.2 GIS and Python 12](#_Toc378521292)

[Section 3: Purpose Statement 13](#_Toc378521293)

[Section 4: Data 14](#_Toc378521294)

[4.1 Data Storage 15](#_Toc378521295)

[Section 5: Methods 16](#_Toc378521296)

[5.1 Data Gathering (Phase I) 16](#_Toc378521297)

[5.2 Interpreting and Digitizing (Phase II) 17](#_Toc378521298)

[5.3 Analysis (Phase III) 17](#_Toc378521299)

[Section 6: Data Analysis 19](#_Toc378521300)

[6.1 Import Modules 19](#_Toc378521301)

[6.2 Calculating Velocity 20](#_Toc378521302)

[6.3 Tracing the Network 25](#_Toc378521303)

[Section 7: Results 29](#_Toc378521304)

[Section 8: Conclusion 30](#_Toc378521305)

[References: 32](#_Toc378521306)

[Table 1 – Manning’s Roughness “coef”ficients 36](#_Toc378521307)

[Appendix A – Velocity.py 37](#_Toc378521308)

[Appendix B 40](#_Toc378521310)

[Appendix C 43](#_Toc378521312)

[Appendix D 44](#_Toc378521313)

[Appendix E 45](#_Toc378521314)

# Section 1: Introduction

Urbanization has increased rapidly over the years and is growing at an exponential rate. Urban areas are densely populated with people and infrastructure so that the communities within the urban environment can thrive. The changes in urban environments, specifically the development of impervious areas, can dramatically change a watershed or wetland, especially ones that are highly agricultural. Most cities develop in areas where there is at least one water source, such as a fresh water source like a river or near an ocean. Urban expansion means a lot of infrastructure which increases the impervious surface.

Impervious surfaces prevent precipitation from absorbing into the ground and into the water table. Thus, when a storm event occurs, water falls to the ground in the form of precipitation and creates runoff causing the water to be transported away from where it would normally absorb. Most cities or counties have poor if any historical data documenting what their regions were like in the past so it has been hard for many environmental groups to provide empirical evidence that impervious surfaces in urban environments have had such a profound effect as they claim (McCauley and Jenkins, 2004). Regardless, cities have had to develop systems for managing the runoff created by the development of buildings and other impervious surfaces like roads. Over time, cities have learned to manage rainfall by using street gutters, channels, and underground storm drain networks.

The urban infrastructure used to transport water under and over ground has evolved over time. Above ground channels using concrete are used to replace streams and rivers, but for areas where urban development requires the water to be transported underground, subsurface drainage infrastructure is implemented. Recent studies have shown that impervious cover and stormwater drainage networks have a profound negative effect on the hydrologic response of the receiving [downstream] waters of the urban environment (Meierdiercks et. al. 2010). The basic elements of a subsurface drainage network consists of: inlets or catch basins along the street gutter or at the bottom of a natural drainage point, connected pipes of various materials and sizes, access structures for maintenance and repairs, and outlet points to disperse of the captured runoff.

Underground infrastructure utilize catch basins that are installed along curb and gutter lines that transport water through small, minimum 18 to 24 inch concrete pipes that attach to the main lines that usually run under public right of ways, such as streets (CALTRANS, 2012; WSDOT, 2010)[[1]](#footnote-1). The materials used for storm drain infrastructure were not designed for that long of a period of time; subsurface drainage within roadbeds is designed for no more than 50 years, and half that when outside the roadbed (CALTRANS, 2012). This is important because some of the current infrastructure in many municipalities was developed more than 70 years ago; thus require a lot of maintenance and can be cause for concern of failure. Age is not the only concern of subsurface drainage networks. An increase in pollutants, natural and manmade, transported into downstream wetlands environments is unintentional offense created by these networks that must also be addressed.

## Land-use influences

Manmade pollutants in urban environments mostly originate from obvious sources such as automobile exhaust, car washes, and illegal disposal of toxic chemicals such as oil, frying grease, and paint. Many believe that these toxins have a relationship to certain land uses within a city and one labels it as buildup and washoff relationships (Pitt et. al. 1995; Charbeneau and Barrett 1998; Hatt et. al. 2004). One study attempted to estimate mass pollutant loading on a macro level with the goal to determine the pollutant contributions from different urban land uses within the watershed of the study area (Ackerman and Schiff, 2003). These studies however weren’t able to provide any information regarding how to find the locations where the pollutants entered the network or identify points of intervention to treat the water before re-entering the natural environment.

## 1.2 Natural Pollutants

Pollutants can also originate from nearby rural areas, mountainous areas, and wild lands that drain in to the city. Also agricultural land use areas outside the city produce pollutants that have a negative effect on [the downstream] receiving waters (Burcher et. al., 2007). Some natural pollutants are created from events such as wild fires that burn vegetation, creating high amounts of carcinogens that are expelled into the air. These particulates then fall to the ground which ultimately most are washed into creeks or the nearby urban stormwater conveyance systems, increasing the amount of pollutants in the water (Stein et. al. 2012). These natural pollutants can change the pH balance of the water causing the life forms to not flourish or provide nutrients within the ecosystem. Not all pollutants originate naturally; many have performed studies that have identified the pollutants within the city.

## 1.3 Pollutant Origins

It is clear now that urbanization has a tremendous impact on the environment. Pollution control is currently one the most important priorities in today’s culture because it emphasizes the need for humans to be more environmentally friendly. Reducing pollution of local water systems is very important; especially in coastal communities because of the impact the pollutants have on the receiving waters and how they affect the marine life (McPherson et. al., 2005; Ackerman and Schiff, 2003). The pollutants originate from multiple sources, but the main three identified so far are waste treatment plant effluents, industrial point-source discharge, and non-point discharge such as improper sewer connections and dumping of hazardous wastes directly into the stormwater system (Schmidt, 1986). Total suspended solids (TSS), nitrates, copper, nickel, lead, and chromium are among the most common water pollutants and are the most commonly measured because of their negative effects on microbial life (Pitt et. al., 1995; Hatt et. Al., 2004; McPherson et. al., 2005). Total suspended solids are any solids saturated in water that will not pass through a 2-micron filter (EPA, 2013). Enough TSS can have poisonous effects on the plants, microbes, and fish that live in the receiving waters. There have been many studies that successfully identified the various types of pollutants which is crucial in increasing the quality of the water entering the downstream environments.

## 1.4 Monitoring Water Quality

Recognizing that there is a problem with urban and rural runoff and its effect on water quality within and downstream of urban environments is a good first step in increasing the water quality. Measuring pollutants in urban runoff is a challenge because there are multiple ways to test water with multiple devices, which is very challenging to standardize when reviewing best practices (Duda et. al., 1982). Many focused on certain land uses that are believed to influence water quality more and analyzed accordingly, leaving out possibilities for illegal connections from residential and vacant land use areas (Charbeneau and Barrett, 1998). Time of year also affects water quality measuring because during times of the year where very little rain falls, most of the water is purely urban runoff which causes the water quality to be very poor. Whereas during the rainy season, most of the water flowing is rain water. Ultimately the potential amount of pollutants is the same (McPherson et. al., 2005). Understanding the overall effects urban environments have had on watersheds and wetlands is hard to estimate. Historical data is rare for most areas, but one study attempted to estimate former depressional wetlands to provide approximate historical data for comparison with data derived from the current local environment of their study and produced mediocre results (McCauley and Jenkins, 2004). Most studies concentrated at a macro level analysis evaluating entire watersheds at once, sometimes focusing on the sub-basins within the watershed to provide more localized analysis (Ackerman and Schiff, 2003; Meierdiercks et. al., 2010). Measuring pollutants and automating system or watershed analysis are only portions of the overall effort in preventing pollution and managing the environment around developed areas.

## 1.5 Watershed Management

Watershed management has evolved over time to include pollution prevention, managing environmental interactions, and now focuses on the construction and maintenance of above and below ground stormwater facilities that will safely transport the water through the city to prevent hazardous conditions for locals in the area (CA-DLRP, 2013). Hazards do not always originate with a sudden storm event however. Regulation has been set at all municipal levels, such as the Los Angeles County Department of Public Works Watershed Management Division, regarding what is legal and safe to drain in to the watershed and the stormwater system (LACDPW-WMD, 2013). Storm drain infrastructure has evolved over the years to accommodate the increasing amount of people and impervious materials in urban areas. Originally, dirt ditches and natural above-ground channels or streams were used to convey water through, around, and out of urban environments (Nelson, 2006). As more and more paved roads dominated the urban environment, street gutters were used mainly to transport water out of the roadways to provide safe transportation for vehicles (Guo, 2000). Not only is the implementation of above ground and underground stormwater systems to provide safety for the people within the urban environment, but it is also to protect the receiving waters downstream of the urban environment so as they will not be adversely effected by peak flows during flood years (Meierdiercks et. al., 2010). Higher volumes of water carry more sediment at a faster rate which can overwhelm the local ecology of the receiving waters with the sediment, volume, and pollutants captured from the urban environment, which is why retention basins are also used to store water during peak flow times (Meierdiercks et. al., 2010).

## 1.6 Study Area

The City of Glendale is one of Los Angeles’ neighboring cities that also shares its border with the Los Angeles River North-West of the City of Los Angeles. The City of Glendale covers 30.7 sq. miles and has a population of 191,719 people (U.S. Census, 2010). Glendale has many businesses throughout the city with many multi-family homes and single-family homes spread through. Flint Peak, of the Verdugo Mountains, splits the city and Verdugo wash flows from La Crescenta through the middle of Glendale until it meets Glendale’s southern border, the Los Angeles River. Glendale is also split transversally by Highway 134 and is bordered by Interstate 5 to the Southwest, Highway 2 through its East side, and Interstate 210 through the Northern end of Glendale (Figure 1).

Although Glendale does not border an ocean, it is within a metropolitan area where all drains lead to the ocean via open channels. Disposing of almost anything besides water into the stormwater system is considered a hazard because it can damage the riparian habitats within the Los Angeles River and ultimately, the beaches and marine life along the coast of Los Angeles County. The most recent policy addressing this is the Clean Water Act of 2001, which created the National Pollutant Discharge Elimination System (NPDES) permit program.

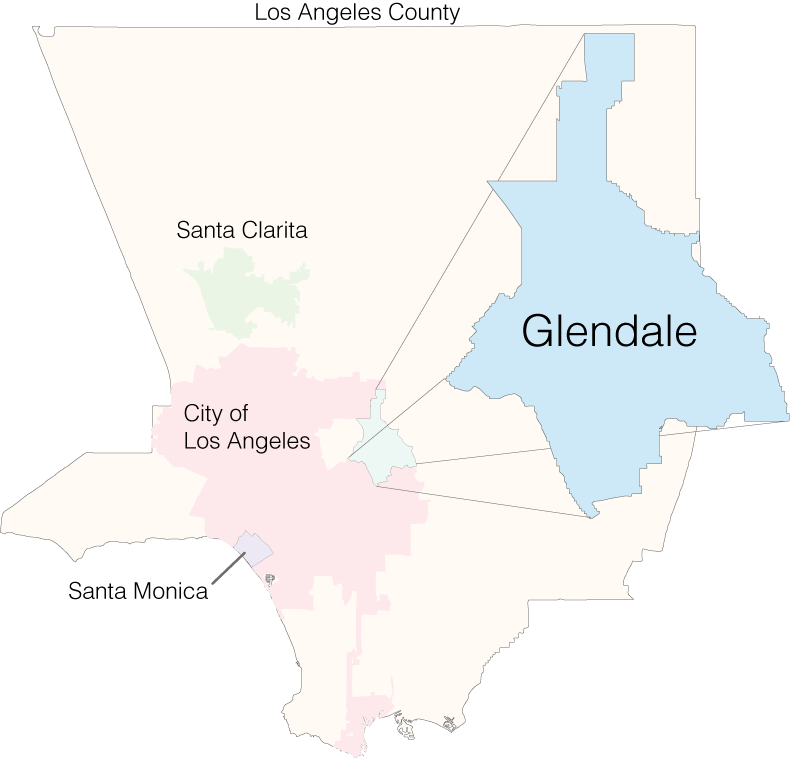


Figure 1- Study Area Glendale, CA

Since hazardous materials are dangerous to the environment, municipalities that govern stormwater systems must enforce guidelines for their citizens that require them to apply for permits to connect any drainage system to the stormwater system. Over the years, there have been policies set at the state and federal levels mandating that local governments across the country must meet minimum requirements in water quality of their stormwater systems. In the Clean Water Act, any river navigable by boat and is accessible to an ocean is classified as a Federal Navigable Waterway (NPDES). Any pollution introduced in to this type of environment results in a penalty against the offending party (NPDES). An NPDES permit requires that municipalities maintain an up to date inventory of all sewer, storm, and potable water facilities. To manage such facilities, municipalities use geographic information systems (GIS) to create a spatial accurate network of the infrastructure and integrate the network with a maintenance management system. Not only does GIS give users the power to map their data, but also it also provides analysis tools to perform such tasks as flow analysis. Some analysis may involve developing custom scripts or programs to perform advanced analysis or automate tasks to aid in the analysis.

There are many factors to be considered when creating a formula that calculates flow times of sub-surface drainage networks. Determining flow times for underground requires very comprehensive GIS data. Not only does the drainage network need to be as spatially accurate as possible, the attribute data of the network feature classes must be as well. Attribute data is tabular data that is associated with a point or line feature that contains whatever information necessary. For this analysis, structural information is the most important data because with key structural information, a model can be constructed to determine how quickly water, or a hazardous spill, moves through the network.

# Section 2: Background

## 2.1 Current Models of Stormwater Analysis

All of the stormwater models found in the research use statistical analysis; whether it is measuring event mean concentrations of pollutants, stormwater runoff mass emissions, annual average flow for different sub-basins, or the amount of impervious surface within a basin or sub-basin (Jewell and Adrian, 1982; Ackerman and Schiff, 2003; Meierdiercks et. al., 2010). Most models are free and readily available to download online with varying degrees of documentation guiding the user how to implement the model and at what degree of precision (Borah, 2011). Since most of the models focus on the macro level, county-wide or greater, of analysis, they utilize shapefiles containing vector lines of existing drainage channels and streams along with digital elevation models (DEMs) to obtain slope and elevation to determine capacity and velocity of the above ground drainage network (Borah, 2011). Some studies worked to develop their own models to better understand the problem they were analyzing as well as to better document the mathematical functions being used to perform the analysis (Ackerman and Schiff, 2003). All these models are very useful for their specific applications, but they all have their strengths and weaknesses.

Such models at the macro-scale do have some values*.* By utilizing existing data of above ground networks, along with DEMs, analyzing the variables associated with above ground flow can be easily automated without having to create or gather a lot of data (Borah, 2011). Also, many of the models are utilized because of the have graphical user interfaces (GUIs) that provide ease of use for analysts that are not familiar with all aspects of the functionality of the model (Borah, 2011). The United States Geological Survey (USGS), United States Army Corps of Engineers (USACE), and United States Department of Agriculture (USDA) developed many of these models, USDA’s TR-55[[2]](#footnote-2), USACE’s HEC[[3]](#footnote-3), and USGS’s PRMS[[4]](#footnote-4), which are free to the public and continuously supported. Other studies developed custom models, ranging from simple and broad to complex and specific, depending on whether they were measuring annual runoff volumes or certain pollutant loading levels (Jewell and Adrian, 1982; Ackerman and Schiff, 2003; McPherson et. al., 2005). Having readily available data and models is very useful, but they also come with their weaknesses.

Such macro-level models leave out the integrated underground stormwater network that captures and transports all the urban runoff that occurs within the city, which has the highest amounts of impervious surfaces and only attribute underground pipes as being a weakness because they may be leaky (Meierdiercks et. al., 2010). Other studies have pointed out before that permitted and illegal connections discharging into the underground system are likely the biggest offenders, but little analysis has been completed to substantiate that claim (Schmidt and Spencer, 1986).

The above studies focused on models that use DEMs with GIS data for above ground drainage infrastructure, like channels and rivers, only to evaluate their watersheds. Digital elevation models are datasets that represent the above ground surface terrain of a specified geographic area (ESRI, 2013). For a highly urbanized city such as Glendale, most of its drainage network is below ground, rendering models that are based on DEMs useless. Most studies focus at the macro level to evaluate pollution within watersheds surrounding a city, but very few have looked at the micro level and considered legal and illegal connections to the stormwater system as having impacts on receiving water conditions (Schmidt, 1986; Ackerman and Schiff, 2003).

## 2.2 GIS and Python

GIS is a powerful technology that can be used as a fully integrated system or as a planning tool and the thousands of different applications of GIS exemplify this power (ESRI, 2012). The leader in GIS software is Earth Science Research Institute (ESRI) and is the software used for this project. GIS is heavily used by government entities to manage all the spatial data involved in managing a city, county, etc. Although GIS tools have been around since the 1960’s, it is constantly advancing in how it can be applied (Coppock, 1991). With recent technological advances in computing power, it is now possible to transfer all the data that could once only be stored on maps and engineering plans into a GIS. When combined with Python, a computer programming language, advanced analysis and automation of tasks can be possible.

To perform the advanced tasks, “libraries” of code have been developed by various developers and software companies. Two libraries used for this project are the “arcpy” library and the “math” library. Arcpy is a custom built Python library that was created by ESRI, which is the software application development company that is the world leader in providing GIS software and geodatabase management applications. This Python module allows the performance of data analysis, conversion, management, and automation of GIS data using either integrated or stand-alone python scripts. The second Python module “math” provides mathematical functions to be performed on non-complex numbers; numbers not including the imaginary unit “i”. This module is necessary because it provides us with the square root function which is part of Manning’s Equation.

# Section 3: Purpose Statement

This study will calculate the flow times for all possible routes of the storm drain network within the City of Glendale. The flow time is the amount of time it takes runoff water to flow through one section of pipe. Generating the data was accomplished by utilizing all available as-built plans of the storm drain infrastructure within the City of Glendale to update the geographic placement and attribute data of the digitized lines that represent the pipes in a geometric network in GIS. Then by using Python, a computer programming language, custom scripts will be written to calculate velocity and flow time for all the active lines that participate in the geometric network.

To calculate the velocity, a custom Python script interprets Manning’s equation for governing flow of water in a conveyance system to calculate, and write into the attributes, the time it takes runoff to flow through each section of pipe. A second script performs downstream traces from each catch basin in the network, selecting the pipe sections of that trace result, then totaling the flow time values of all selected pipes to determine the time it takes water to flow from each catch basin to the end of the network. These calculations will be done for the dry season conditions and wet season conditions because of their variance in flow volume and speed.

# Section 4: Data

The data used for this project was acquired from LACDPW (Los Angeles County Department of Public Works) where a team of engineers, surveyors, and GIS professionals in 2009 started updating the data to reflect which drains LACDPW own and maintain; as well as correct the geographic geometry using 4 inch pixel resolution aerial imagery as well as the original as-built construction plans of all the projects that built sections of the storm drain network.. The as-built plans were scanned and are available in databases at the City of Glendale as well as at LACDPW. The aerial imagery used was acquired from LARIAC (Los Angeles Region Imagery Acquisition Consortium); a group of municipalities, professional groups, and universities that collectively pays for bi-annual, high-resolution imagery of Los Angeles County. All the data is hosted in ArcSDE (Spatial Database Engine), which translates the spatial data between the database and the client user. There are 32 feature classes stored in a feature dataset, which allows for trace functionality to be possible like the geometric network.

## 4.1 Data Storage

The GIS database model is designed to accommodate different functionalities such as a geometric network. The basic design of every geodatabase contains one or more feature classes, which is a single table containing data of the same type (point, line, or polygon), one or more stand-alone tables (does not contain/display geometry), or one or more feature datasets which contain one or more feature classes and or tables. A geometric network is a set of connected points and lines along with connectivity rules that model the real life behavior of the network it represents (ESRI, 2012). The line features are referred to as edges that are connected by junctions at points of intersections with one or more other edges.

All polyline features participating in the geometric network can be configured to be either simple or complex edges. Simple edges allow only for connectivity rules to be applied at either end of the polyline. A complex edge would include any intersecting polyline with midspan connectivity (ESRI, 2012). This means that when a trace function is to be performed, if another edge were connected between either ends of the original polyline, the trace would not include the edge connected midway. The edges also contain all the attribute information of the polyline feature classes that represent the stormwater pipe.

# Section 5: Methods

## 5.1 Data Gathering (Phase I)

The first phase was the gathering of legal documents, such as as-built plans and storm drain project transfer records. As more businesses and homes were built, the more impervious surfaces were created. To manage the increase in urban runoff due to the increase in impervious surfaces, civil engineers designed storm drain systems to capture and transport the water safely either above or below ground. The systems were drawn out to scale in an engineering plan which was denoted with information regarding the topography of the landscape, existing infrastructure and landmarks such as roads, as well as the location of where the project should be built and the specifications of the materials and dimensions to be used.

The plans were developed by the city’s engineering department, the county’s design engineers, or by a private contractor. Regardless of who designed it, these plans are reviewed and approved by the City Engineer throughout the process; at the time of proposal, construction and completion; upon completion and approval is when a project and the plans are updated and labeled as “As-Built”. These as-built plans must be kept on record by the city and county to be used to provide information about the underground infrastructure and where it is located. Sometimes, the city may want to relinquish its ownership and maintenance duties for the system and applies to transfer the ownership and maintenance to the County of Los Angeles’s Flood Maintenance Division; in which case, the county would then assume all responsibility of the drainage system.

## 5.2 Interpreting and Digitizing (Phase II)

To accurately represent the storm drain network digitally, the second phase involves interpreting the as-built plans, and transfer records when necessary, and digitizing the network in a GIS as accurately as possible using the highest resoultion orthoimagery available. Interpreting the as-built plans, along with the imagery, the features must be digitized in their correct spatial location and their attribute tables populated with the dimensions such as height, width, type, material, and slope as well as ownership information. There are two main components that share top priority when editing; ownership and geographical location. In GIS, the location of the points and lines will determine the length of the segments of pipes, which influences the slope. Without length, slope cannot be determined and without slope, velocity cannot be calculated. The ownership is important because legally the party responsible must maintain their system to prevent failure, but also they do not want to maintain a system that is not theirs to prevent liability if they accidentally cause damage.

## 5.3 Analysis (Phase III)

Manning’s Equation is an engineering equation developed by Robert Manning and is a formula that captures the mean velocity of gravity-driven, fully developed flows of water in rough open channels as well as underground pipe infrastructure (CALTRANS, 2006). By using Manning’s Equation (seen below) and the proper roughness “coef”ficients (Table 1)[[5]](#footnote-5) for the material of the system, the necessary parameters can be determined to calculate velocity for underground stormwater infrastructure (Lindeburg, 2013).

Q = Flow Rate (ft3/s)

V = Velocity (ft3/s)

A = Flow Area (ft2)

*n* = Manning’s Roughness “coef”ficient

R = Hydraulic Radius (ft)

S = Channel Slope (ft/ft)

Equation 1 - Manning's Equation

Manning’s Equation calculates the flow rate of water by dividing a constant “1.49” (for imperial measures) with the Manning’s Roughness “coef”ficient “*n”* (Table 1) of the material of the system, multiplied by the flow area “A”, multiplied by the Hydraulic Radius “R”, raising that value to the power of 2/3 and then multiplying the square root of the slope “S” of the infrastructure.

To automate the use of Manning’s Equation within a GIS, a python script will translate the equation for velocity into steps that the computer can calculate based on the attribute data in the pipe segments of the geometric network of storm drains. This script can be scheduled to execute periodically, or executed on demand, as the attributes and geographic components of the network are updated because every update can help tune the network to be as close to reality as possible. The next step will be to run a script that will total the flow times of each pipe of a downstream trace for every catch basin in the network. Then the script will write the total flow time in to the catch basin used as the origin of the flow to identify the potential flow time downstream. If given the option of an output workspace, the script will export a single line representation of the traced path.

# Section 6: Data Analysis

The analysis performed for this project is conducted almost entirely using custom Python scripts. These scripts serve as a detailed outline for the workflow process necessary to produce the data accurately and efficiently. By performing the analysis through custom Python scripts, it creates built-in documentation, as well as allows for this analysis to be performed repetitively and if needed, it can be automated. This analysis was performed using two Python scripts, Velocity.py and Trace.py (Appendix C and D). Flow diagrams were created for each of these scripts to document the logic of each script as it processes and can be seen in Appendix A and B.

To explain the process of analysis, the scripts will be explained in the order of their logic. This means that tools are not executed in the order they appear line by line in the scripts found in Appendix C and D. Variables denoted as “local” means they are only relevant in the block of code or loop in which they are created. If a variable is relevant outside of that block or loop, it is known as a “global” variable.

***6.1 Interpreting Coded Values***

Before running the scripts, understanding the coded domain values of the data is important to complete. The dataset obtained from LACDPW used coded domain values to populate the materials for the pipes and channels. The code is a numeric value that is associated with a textual value and is stored as a table in the database. It’s important to update the arrays “materialList”, “materialPlaticsList”, and “materialErrorList” in the Velocity.py script when executing against a dataset that may have different coded values or none at all. See Table 2 for the coded domain value table for LACDPW’s storm drain network.

## 6.2 Import Modules

There are many different Python modules needed to complete the processing and calculations. The libraries are arcpy, math, sys, os, and traceback. The arcpy module is a developed by ESRI and deployed with their ArcGIS for Desktop (v10.0 and above) desktop application. It provides access to most of the tools that are used in the desktop application. The math module provides functions to perform advanced math equations. In this script, it is required to have access to the square root and exponent functions. The os and sys modules provide tools for creating files and folders, and the traceback module provides tools for reporting error information that may occur at the system level.

## 6.3 Calculating Velocity

Velocity.py calculates the velocity of every storm drain pipe segment represented as a polyline feature that participates in the geometric network by evaluating the row in the table that represents the path in the database; row is synonymous with a pipe segment and path is synonymous with table. The velocity.py script begins by importing the necessary modules and defining the workspace; which is the directory containing a file geodatabase with the geometric network data included. Next the script enters into a try block; this is where most of the script’s processing is controlled. The try block is used to catch errors that may occur in the processing. If any errors occur the script will immediately jump into the following except block to execute error procedures before ending the script. Once in the try block, a series of variables are initialized that contain arrays (lists) of related data used throughout the script. The first list variable is the “materialList” which contains the coded domain values of the valid materials used to construct the storm drain pipes that use the same Manning’s “coef”ficient “0.012”. The second list variable “materialPlasticList” contains the list of coded values that use “0.009”. The third list variable “materialErrorList” contains the list of coded values of all the invalid materials or features without a value entered represented by the Python keyword for null “None”. The fourth list variable “newFields” contains names for fields to be added to traced polyline feature classes. The names are “Velocity\_fps” and “FlowTime\_secs” and they will represent the velocity in feet per second and the time in seconds it takes for water to flow through a pipe segment. The fifth list variable contains a list of all the polyline feature classes that will have the fields added to them and their corresponding values calculated.

Next, a “for” loop is started that loops through the paths list and creates a temporary variable called “fields” that contains all of the fields of one of the paths. Then it loops through the “newFields” list, checks to see if the any of the “newFields” are not in the current “fields” list. If not, the script adds the field to the path using the arcpy function “AddField” and notifies the user the field was added. Following the loop, a variable called “edit” uses the arcpy Data Access function “editor” that returns an object that will allow editing to be started and stopped. Next, an edit session is initialized and an operation is started. This is necessary because if for any reason an error occurs during processing, the script will skip to the “except” block and stop the edit session without saving changes.

Once the edit session is started, another loop of the paths is started and a new list variable is created called “fields”. The “fields” variable contains a list of only the fields that are necessary for processing and calculation. These fields are “MATERIAL”, “DIAMETER\_HEIGHT”, “SLOPE”, “Velocity\_fps”, “Flowtime\_Secs”, “SHAPE\_Length” and “OID@”. An “if” statement is used to check if the path is “OpenChannel” and if it is, the field “SOFT\_BOTTOM” is appended to the fields list. This field holds “Y/N” values to designate whether or not the bottom of the channel is “soft”, meaning it is earthen. Next a “with” statement is used to initialize an arcpy Data Access Update Cursor on the current path value of the paths list, with the fields list variable passed in as its second parameter, and the cursor object is set to the variable named rows. Then a “for” loop is started on the cursor object rows to initialize the cursor to begin iterating through all of the rows representing line segments of the polyline path. Once in the “for” loop, an “if/else” block is used to check which path is being updated. If it is the “OpenChannel” path, then the variable “edge” is initialized with the “channelObj” by passing in the path and row objects. If it is not “OpenChannel”, the edge variable is initialized with the “pathObj” by passing in the path and row objects.

The “channelObj” and “pathObj” are static objects that are declared globally before the majority of the code, much like functions. Objects are useful because they have attributes that are predefined and populated, updated, or even deleted anywhere in the code. It also makes the code easier to read and use because the object’s attributes are clearly defined so the user knows. Both objects require the same arguments, a string of the “path” and a row object. The difference is the “channelObj” has an additional attribute “softBtm” that represents the “SOFT\_BOTTOM” field that “OpenChannel” row object has. Once the edge variable is initialized with one of the objects, it is then passed in to the “checks” function.

The “checks” function is defined globally before the try block. The purpose of the “checks” function is to perform a series of checks on some of the variables for validity. The “checks” function requires one argument, the edge variable with either the “pathObj” or “channelObj”. In the “checks” function a new variable called “coef” is initialized and is initialized with “0.0” to define itself as a floating point variable. Next an “if” statement is used to check the slope attribute of the edge variable value to see if it is zero or “None” (null, meaning no value). If it is, set the slope attribute is updated with the value 0.03, which is the most common slope value. If the value is neither zero nor “None”, then the slope attribute will continue to hold its original value. Another “if” statement is used to then evaluate the height attribute for the values zero, “None”, or if it is greater than or equal to “999”, which is sometimes used as a value to denote the absence of the known value. Next the material attribute is passed into a series of “if” and “elif” statements. “Elif” statements act as switch or case statements in other languages which means that whichever if or “elif” statement evaluates to true, the expressions or conditions within that true statement will be expressed and any remaining statements will be skipped (Python Docs, 2013). The first “if” statement checks if the material attribute is in the list variable “materialList” and if it is, the “coef” variable is set to “0.012”. Next the material attribute is checked in an “elif” statement to see if it is in the “materialCorrugatedList” and if it is, the “coef” variable will be set to “0.22”. Another “elif” statement is used to see if the material attribute is in the “materialPlasticList” and if it is, the “coef” variable is set to “0.01”. A third “elif” statement is used to see if the material attribute is in the “materialEarthList” and if it is, the “coef” variable is set to “0.025”. A fourth “elif” statement checks if the material attribute is equivalent to the value “13” and if it is, the “coef” variable is set to “0.01”. The last “elif” statement checks if the material variable is in the materialErrorList list and if it is the error function is called. The “error” function takes in the “path”, “oid”, “material” attributes and a string “material” and will report back to the user the field the value is set, the value itself, and the unique identifier for the row the value is set so that the user can identify exactly where the invalid value occurred. It also updates that row object with zeroes in the “FlowTime\_secs” and “Velocity\_fps” fields so no invalid data will be populated. Lastly the “velocity” function is called and is passed the checked “edge” object and the “coef” variable. The length attribute was not checked because it is generated by digitizing the paths in the GIS and they cannot be modified by the user.

The “velocity” function begins be initializing a variable called velocity and the results of the programmatic version of Manning’s Equation; utilizing the “height” and “slope” attributes “edge” object passed in to the function as well as the “coef” variable passed in to the function . Next a variable called time is initialized and contains the result of the equation which divides the “length” attribute of the edge object by the results in the “velocity” variable. Next the index values “3” and “4” of the row object are set with the values “velocity” and “time” respectively. Lastly a statement prints to the user the velocity and time set for the unique id for the row evaluated. Once the “velocity” function completes, it returns back to the end of the “for” loop where the “updateRow” function is called on the “row” object to commit the changes into the feature class.

The script has successfully completed its first iteration through the loop of rows in the path being evaluated. It will then perform the process again beginning at the start of the “for” loop by checking if the path is “OpenChannel”, initializing the appropriate object, passing that object into the “checks” function. Upon successful completion of the checks, it continues on to the “velocity” function, whereupon calculating velocity and flow time for the row. The script will perform this process for every row in the table of the path being evaluated. Once all rows have been evaluated for one path, the process will repeat for the next path in the paths array. Upon successful completion of all the rows of all the paths, the “stopEditing” function is called on the edit object and is passed the value “True” to save all changes and stop the edit session.

If any error were to occur in evaluating any of the values of any of the rows of any of the paths, the script will skip out of the try block and straight in to the except block. The “except” block is used to stop any processes, reverse any changes, and report error messages to the user. The first step in the except block of this script is to call the “stopEditing” on the edit object and pass the value “False” to stop the editing session and not save any changes. A message is printed to the user notifying them this occurred. Next two variables “tb” and “tbinfo” gather the python traceback objects from the “sys” and “traceback” modules. These two variables are then concatenated into a string (a text object) message set to another variable called “pymsg”. A fourth variable is created called “msgs” and contains a concatenation of predefined text with any messages returned from the arcpy module. Lastly, a print statement is called to report all the messages to the user.

The last section of the script is the “finally” block which is used simply to print a statement to the user that the script is complete. The results of running this script is that all the segments of all the polyline feature classes passed in as paths in to the script will have a velocity and flow time calculated for them. These values are useful for understanding the dynamics of each piece in the system, but are even more useful for the second script used to evaluate the system. The velocity.py script takes 18 to 19 seconds to complete for all four polyline feature classes. The total time is dependent on the number of feature classes and the number of features within each feature class.

## 6.3 Tracing the Network

Now that the velocities and flow times for the paths of all the feature classes have been calculated, the second script “trace.py” will perform a downstream trace from every catch basin in the network. First, the script imports four python modules: arcpy, os, sys, and traceback. A variable called “output” is initialized with a string of the directory the user wants the output file geodatabase located. An “if” statement then evaluates if the “output” variable has a value populated. If there is a value, then an “if” statement is used with the “os.path.exists” function to check if the output file geodatabase already exists. If it does exit, the script deletes the existing output file then a new empty file geodatabase is created. The file geodatabase is created using the arcpy tool “CreateFileGDB” which requires the path to a folder and the name of the file geodatabase to be created. The output variable contains both the path and the name of the file geodatabase to be created. Two variables are created called “path” and “name”; each split the output variable on the last reverse oblique (aka “backslash” or the “\” symbol) and the first part is set to “path” and the last part is set to the “name” variable. The variables are then passed in to the “CreateFileGDB” function to create the output file geodatabase.

Once the output file geodatabase is created, a new variable called “workspace” is initialized which contains the path to the data in another file geodatabase. The arcpy workspace environment is initialized with the “workspace” variable and the “overwriteOutput” option set to “True” to ensure no conflicts in data output can cause the script to fail. Next the script enters in to the try block and initializes a series of variables. The first variable is “sdnNet” which is a concatenation of three elements: the “workspace” variable, the feature dataset the geometric exists in with double reverse obliques at the beginning and end “\\SDN\\”, and the name of the geometric network of the storm drain network “STORMDRAINNET\_NET”. The second variable is “flags” which contains “CatchBasin” which is the point feature class being used as the origin of the trace of the geometric network. The third variable is the “fields” variable which contains the output of a list comprehension performed on the “flags” variable to obtain a list of the fields it contains. The fourth variable is the “newField” variable which is a list of only one string value “FlowTime”; which will be used to write the output total flow time of the trace that originated at that catch basin. A “for” loop then loops through the “newField” list variable to check if it’s value “FlowTime” is in the “fields” list variable and if it’s not, the script adds “FlowTime” to the “CatchBasin” feature class.

Next the edit object is initialized with the input “workspace” variable and the “startEditing” and “startOperation” functions are called to begin an editing session. A fifth variable “flow” initialized with the string value “WITH\_DIGITIZED\_DIRECTION” is passed in to the function “SetFlowDirection\_management” along with the “sdnNet” variable. This function with the parameters given will insure that the flow direction of the network is set to follow the digitization methods used to create the network, which were to draw the segments downstream. A sixth variable is initialized called “flagFields” as a list of the fields “OID@” and “FlowTime” that will be used by the update cursor. The “da.UpdateCursor” is then initialized using a “with” statement, is passed the “flags” and “flagFields” variables and is set to the object “rows”. Next a “for” loop of the “rows” object will iterate through all the rows of the “CatchBasin” feature class by returning a temporary “row” object to be used for enumeration.

Once in the “for” loop, four variables are initialized. The first variable “oid” holds the value from the first field in index of the “rows” object; which is the “OBJECTID” value of the “CatchBasin” feature class. The second variable “newNet” holds a concatenation of “SDN\_Net” and the oid value casted as a string. The “newNet” variable will be used as the name of the temporary output of the trace tool. The third variable “flag” holds a concatenation of “flag” and the oid value casted as a string. The “flag” variable will be used as the name of the temporary input flag for the trace tool. The fourth variable “exp” holds the beginning of a sql expression ‘”OBJECTID” = ‘ concatenated with the oid value of the current catch basin casted as a string. The “exp” variable will be used in the “MakeFeatureLayer” tool to create a layer of a single catch basin to be used for tracing.

Next the “MakeFeatureLayer\_management” function is used to create an in memory representation of only one catch basin to be used as a flag in the trace tool. The function is passed the “flags”, “flag”, and “exp” variables which identify the input feature class to create a feature layer from, the name of the feature layer being created, and an optional SQL expression respectively. The “TraceGeometricNetwork\_management” tool is then used to perform the trace of the geometric network. The trace function is passed the input geometric network (“sdnNet”), the name of the output group layer (“newNet”), the input flag to start the trace from (“flag”), the trace method (“TRACE\_DOWNSTREAM”), and another 14 optional variables that in this usage, are given the tool’s default values. The result of the function is a selection of all valid feature classes of the input geometric network (“sdnNet”).

A series of five variables are then initialized: “gravityMain”, “lateralLine”, “openChannel”, “culvert”, and “psuedoLine” all the names of the feature classes that could have been selected in the trace. The first four variables are set in a list variable called “pathList” to be used for enumeration to get the flow times from each selected path of each feature class. Another variable “mergeList” is initialized as an empty array and the last variable “total” is initialized with the value “0.0”. These variables will be populated with the results of the “pathList” enumeration.

Next a “for” loop iterates the “pathList” variable and initializes an object called “path”. In this loop the variable “results” is initialized with the “calcFlow” function which is passed the two required parameters: “output” and a concatenation of the the “newNet” variable and the “path” object. In the “calcFlow” function, a local variable “time” is initialized with the value “0.0”, which will hold the total time for the “path” object passed in. Next a variable “mergePath” is initialized to hold the results of a ternary operator that checks if the “path” object passed in has records or not using the “arcpy.GetCount” function. If the path does, “mergePath” holds the value “True”; if it does not have any records, it holds the value “False”. An “if” statement then evaluates “mergePath” for the value “True” and if it is, the “arcpy.da.SearchCursor” function is initialized with the “path” object and single value array “FlowTime\_secs” as the field to be returned by the “rows” object.

A “for” loop then enumerates the “rows” object and creates a single “row” object. A variable “flow” is initialized with the value from “FlowTime\_secs”, is checked that it is not “None” or a space. If it is neither, the “time” variable is incremented with the value in the “flow” variable and the “for” loop then enumerates to the next “row” object until all the rows have been evaluated. Lastly the “return” statement is called, passing back the values held in the “time” and “mergePath” variables. If the a path had rows and flow time values, the values passed back in the array will be the total of the flow times and the value “True”. If the path did not have rows, the array passed back will be the original value of “time” and the value False.

Once the “calcFlow” function is complete and values passed back to the “results” variable, the “total” variable is incremented with the first value of the “results” array. An “if” statement then evaluates the second value of the array and if the value is “True” the “path” that was enumerated is appended to the “mergeList” list. At this point the “for” loop of the “pathList” will iterate the list until all the “path” objects have been enumerated. Once that is completed, the script will then print to the user “total tome is “ and the value of the “total” variable, set that value to the “FlowTime” field of the “CatchBasin” feature class, and use “updateRow” to write the value in to the table.

If the user defined a value for the “output” variable and the “mergeList” variable is not empty, the output process begins. A variable “tracePath” is initialized with a concatenation of the “output” variable, double reverse obliques “\\”, the string “tracePath\_”, and the “oid” variable casted as a string. A second variable “pseudoRecords” is initialized and holds the results as an integer of the record of the of the “pseudoLine” feature class. If it has records, the variable “pseudoLine” is appended to the “mergeList” variable. This is done to make sure empty feature classes aren’t attempted to be merged into feature classes with selected records. Next, the “createOutput” function is called and is passed the “tracePath”, “output”, “mergeList”, and “newNet” variables.

In the function “createOutput” the “Merge” tool is called with the “mergeList” variable passed in as first parameter and “tracePath” variable passed in as the second parameter. The “mergeList” variable holds the list of paths to merge into a single feature class and it is named with the value of the “tracePath” variable. Next the “Dissolve” tool is called and is passed the “tracePath” variable as the paths to be dissolved, a concatenation of the “tracePath” variable and “\_dissolved” for the output name, an octothorpe to denote no value for the dissolve field, another octothorpe to denote no values for the statistics fields, the string ‘SINGLE\_PART’ to denote multipart features are not allowed, and finally ‘DISSOLVE\_LINES’ to denote that features are to be dissolved into a single feature. Next the “Delete” tool is called and is passed the “tracePath” variable to delete it since it is temporary output. Once the “createOutput” function has completed, a variable “delete” holds an array of temporary variable outputs “flag” and “newNet” is initialized and iterated. For each item in the “delete” array, the item is passed into the “Delete” function to be deleted. Lastly the “stopEditing” function is called with the value “True” to save the changes.

Once the script has completed successfully in the “try” block or has completed execution of the “except” block, the “finally” block is called. In this script the “finally” block contains a single print statement notifying the user that the script is completed. The trace.py takes up to 6 hours to complete when not generating output for the 4 feature classes provided. When generating output trace lines, the script takes up to 24 hours. The total time is dependent on the number of feature classes and the number of features within each feature class.

# Section 7: Results

The first script calculates the potential velocity and total flow time for the pipe segment based on input engineering specification and geographic measurements stored in each segment of the geometric network. Once calculated, the script then writes the velocity and flow time values into two columns within the feature class being calculated, “Velocity\_fps” and “FlowTime\_secs”. The results of successful completion are the velocity and flow time values calculated for each pip segment of the network. The accuracy of the velocity is dependent on all four specifications. The most common error or inaccuracy is the slope measurement. The measurement is often hard to capture because it cannot be derived using GIS tools for underground structures and it is not always available on as-built construction plans like the material and height/diameter measurements. The length measurement is always available because it is auto-generated by the GIS.

The results of the first script are used in the second script which automates the tracing of every possible flow path of the storm drain system. For each path traced, every segment along the path is selected; the flow time for each path is totaled and then written into a field called “FlowTime”. The results of the second script create a new output feature class representing the traced path with the flow time information and flow geometry. The flow time information is also populated in the catch basin used as the origin of the trace. As an optional output, the trace path can be exported into an output file geodatabase with the OBJECTID of the origin inlet point feature included in the name (i.e. “tracePath\_127”). The accuracy of the trace paths is dependent on the quality of the geometric network. Gaps between lines that are supposed to be connected, corrupted junction features, and lines that are not split on junctions result in incomplete trace paths or trace paths with portions of features that should not be included in the network. To improve the output features, these errors should be corrected.

# Section 8: Conclusion

As population increases, so does impervious surface in and around cities which increases the need to manage the runoff created whether it is natural or manmade. With advancements in environmental research and understanding, geographic information systems (GIS) has helped to better manage infrastructure by providing locational accuracy as well as analysis capabilities to understand what is happening in underground infrastructure. Using engineering drawings and collaboration with maintenance crews, storm drain infrastructure can be digitized into GIS to allow them to perform their preventative maintenance and emergency response services. Also it helps predict what the wear status of the system is by factoring in age and usage. The analysis capabilities allow organizations to predict what parts of the system needs maintenance or replacement. It can help in emergency situations to predict the volume of a flow, its path, and how long it takes to reach certain points.

These capabilities also help monitor water quality as urban areas change. Monitoring water quality requires understanding where natural and manmade pollutants originate and how they will flow through the storm water system. This can help responsible parties plan for how they can better manage these pollutants within the system or at the systems outfalls. Not only is the water quality important, but these tools can be used in watershed management and urban planning to help understand how additional structures and impervious surface can impact the current system. Other tools and models provide similar capabilities with above ground, open surface drainage infrastructure; but these tools can extend the analysis capabilities by capturing the underground infrastructure. These scripts are highly adaptable and can be used for many other systems, such as sewage and potable water systems.

Custom programming like this is often necessary to automate workflows specific to certain needs, but can only solve so much of the problem. Data accuracy is the most important part of analysis. For storm drain networks, gathering the “as-built” engineering drawings is crucial to acquiring the exact specifications of all elements of the storm water infrastructure, aside from physically inspecting the whole system in person or robotically. By understanding the engineering specifications of the storm water infrastructure and the elements of those specifications necessary to calculate volume and velocity, the scripts are able to automate the calculation of these values.

# References:

Ackerman, Drew and Schiff, Kenneth. 2003. Modeling Storm Water Mass Emissions to the Southern California Bight. *Journal of Environmental Engineering*. Vol 129 pp. 308-317. American Society of Civil Engineers.

Borah, Deva K. 2011. Hydrologic procedures of storm even watershed models: a comprehensive review and comparison. *Hydrological Processes*. Vol. 25. pp. 3472 – 3489. wileyonlinelibrary.com

Burcher, C. L., Valett, H. M., and E. F. Benfield. 2007. The Land-Cover Cascade: Relationships Coupling Land and Water. *Ecology*. Vol. 88, No. 1. pp. 228 – 242. Ecological Society of America.

CA-DLRP (California Division of Land Resource Protection). 2012. <http://www.conservation.ca.gov/dlrp/wp/Documents/California%20Watershed%20Program.pdf>

CALTRANS. 2006. Highway Design Manual. Page 7 <http://www.dot.ca.gov/hq/oppd/hdm/pdf/english/chp0860.pdf>

CALTRANS, 2012. Highway Design Manual.

<http://www.dot.ca.gov/hq/oppd/hdm/pdf/english/HDM_Complete_06-21-13.pdf>

Charbeneau, Randall J. and Barrett, Michael E. 1998. Evaluation of Methods for Estimating Stormwater Pollutant Loads. *Water Envrionment Research*. Vol. 70, No. 7. pp. 1295 – 1302. Water Environment Federation.

Coppock, J.T., Rhind D.W. 1991. The History of GIS. *Geographical Information Systems: Principles and Applications*. Vol 1.1 pp. 21-43.

Duda, Alfred M., Lenat, David R., and Penrose, David L. 1982. Water Quality in Urban Streams: What We Can Expect. *Journal (Water Pollution Control Federation)*. Vol. 54, No. 7. pp. 1139 – 1147. Water Environment Federation.

EPA (Environmental Protection Agency) 2013. <http://water.epa.gov/type/rsl/monitoring/vms58.cfm>

ESRI. 2012. [www.help.arcgis.com](http://www.help.arcgis.com)

ESRI. 2013. <http://support.esri.com/en/knowledgebase/GISDictionary/term/DEM>

Guo, James C. Y. 2000. Street Stormwater Storage Capacity. *Water Environment Research*. Vol. 72. No. 5. pp. 626 – 630. Water Environment Federation.

Hatt, Belinda E., Fletcher, Tim D., Walsh, Christopher J., and Taylor, Sally L. 2004. The Influence of Urban Density and Drainage Infrastructure on the Concentrations and Loads Pollutants in Small Streams. *Environmental Management*. Vol. 34, No. 1. pp. 112 – 124. Springer Science and Business Media Inc.

Jewell, Thomas K. and Adrian, Donald Dean. 1982. Statistical Analysis to Derive Improved Stormwater Quality Models. *Journal (Water Pollution Control Federation)*. Vol. 54, No. 5. pp. 489 – 499. Water Environment Federation.

LACDPW-WMD (Los Angeles County Department of Public Works – Watershed Management Division). 2013. <http://ladpw.org/wmd/>

Lindeburg, Michael R. 2013. Mechanical Engineering Reference Manual for the PE Exam. 13th Edition. Professional Publication, Inc. (PPI).

McCauley, Lisa A. and Jenkins, David G. 2004. GIS-Based Estimates of Former and Current Depressional Wetlands in an Agricultural Landscape. Vol. 15. No. 4. pp. 1199 – 1208. Ecological Society of America.

Meierdiercks, Katherine L. et. al. 2010. Heterogeneity of Hydrologic Response in Urban Watersheds. *Journal of the American Water Resources Association*. Vol. 46, No. 6. pp. 1221 – 1237. American Water Resources Association.

McPherson, Timothy N., et al. 2005. Trace Metal Pollutant Load in Urban Runoff from a Southern California Watershed. *Journal of Environmental Engineering*. Vol 131 pp. 1073 – 1080. American Society of Civil Engineers.

NPDES. National Pollutant Discharge Elimination System. U.S. Environmental Protection Agency. <http://cfpub.epa.gov/npdes/>

Nelson, Peter A., Smith, James A., and Miller, Andrew J. 2006. Evolution of channel morphology and hydrologic response in an urbanizing drainage basin. *Earth Surface Processes and Landforms.* Vol. 31 pp. 1063 – 1079. www.interscience.wiley.com

Pitt, Robert, Field, Richard, Lalor, Melinda, and Brown, Michael. 1995. Urban Stormwater Toxic Pollutants: Assessment, Sources, and Treatability. *Envrionment Research*. Vol. 67, No. 3. pp. 260 – 275. Water Environment Federation.

Python Docs. 2013. <http://docs.python.org/>

Schmidt, Stacy D. and Spencer, Douglas R. 1986. The Magnitude of Improper Waste Discharges in an Urban Stormwater System. *Journal (Water Pollution Control Federation).* Vol. 58, No. 7. pp. 744-748. Water Environment Federation.

Stein, Eric D. et. Al. 2012. Stormwater Contaminant Loading Following Southern California Wildfires. *Environmental Toxicology and Chemistry*. Vol. 31. SETAC Press.

WSDOT Hydraulics Manual M 23-03.03. June 2010. Page 6-17

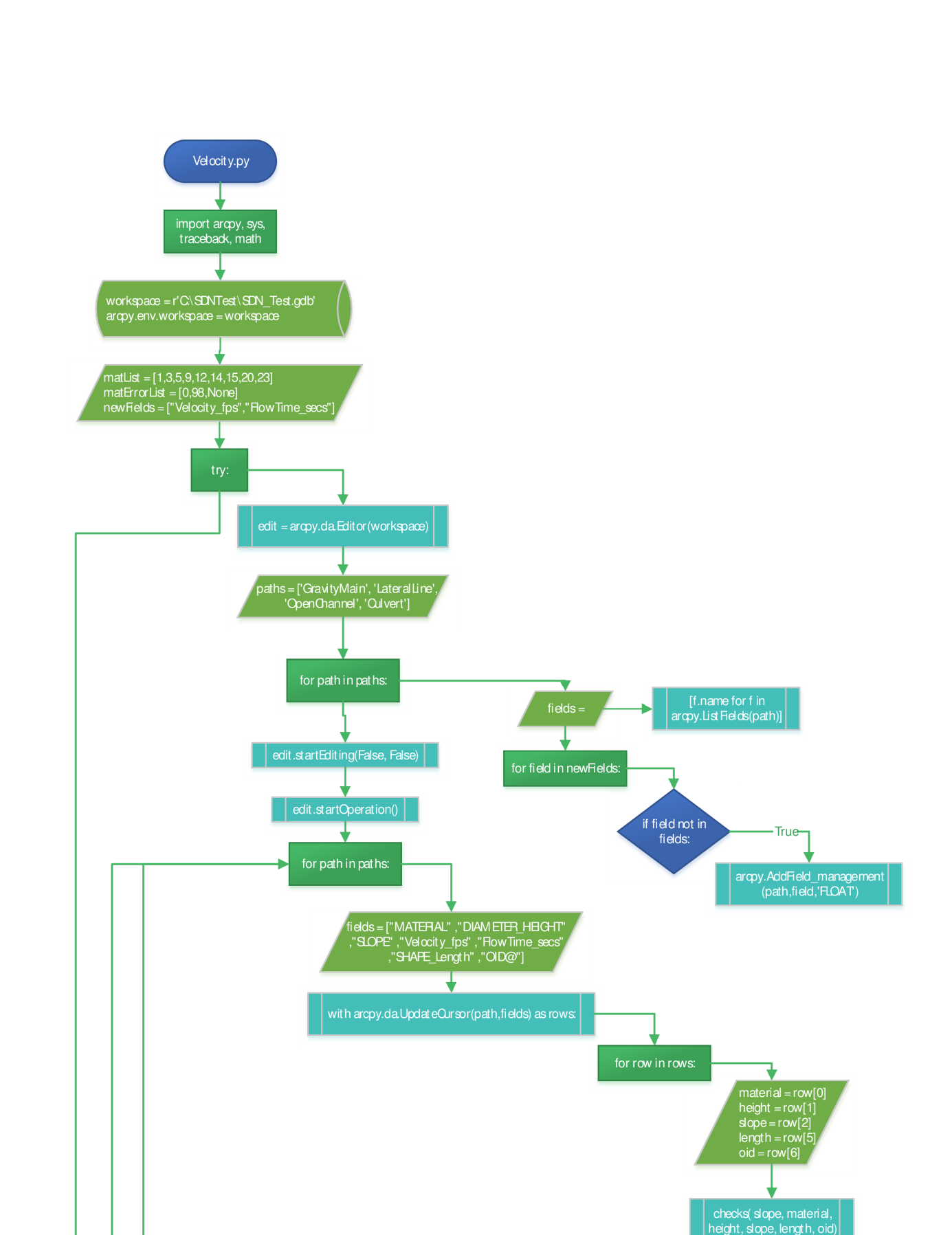
# Table 1 – Manning’s Roughness “coef”ficients

****

# Table 2 – Coded Domain Values

|  |  |
| --- | --- |
| Code | Value |
| -9 | Error? |
| 0 | Not Coded |
| 1 | Reinforced Concrete Pipe (RCP) |
| 2 | Plastic Pipe |
| 3 | Reinforced Concrete Box (RCB) |
| 4 | Corrugated Metal Pipe (CMP) |
| 5 | Reinforced Concrete Arch (RCA) |
| 6 | Cast/Ductile Iron Pipe (CIP) |
| 7 | Improved Channel |
| 8 | Corrugated Steel Pipe (CSP) |
| 9 | Concrete Pipe |
| 10 | Acrylonitrile-Butadiene-Styrene (ABS) |
| 11 | Polyvinyl Chloride (PVC) |
| 12 | Steel Pipe |
| 13 | Vitrified Clay Pipe (VCP) |
| 14 | Unreinforced Concrete Pipe |
| 15 | Asbestos Cement Pipe |
| 16 | Polyethylene Liner |
| 17 | Techite |
| 18 | Dirt Channel |
| 19 | Dirt Swale |
| 20 | Brick |
| 21 | Cured-In-Place Pipe Liner (CIPP) |
| 22 | High Density Polyethylene Pipe (HDPE) |
| 23 | Reinforced Cement Concrete(RCC) |
| 24 | TRUSS PIPE |
| 98 | ? |
| 99 | Other |

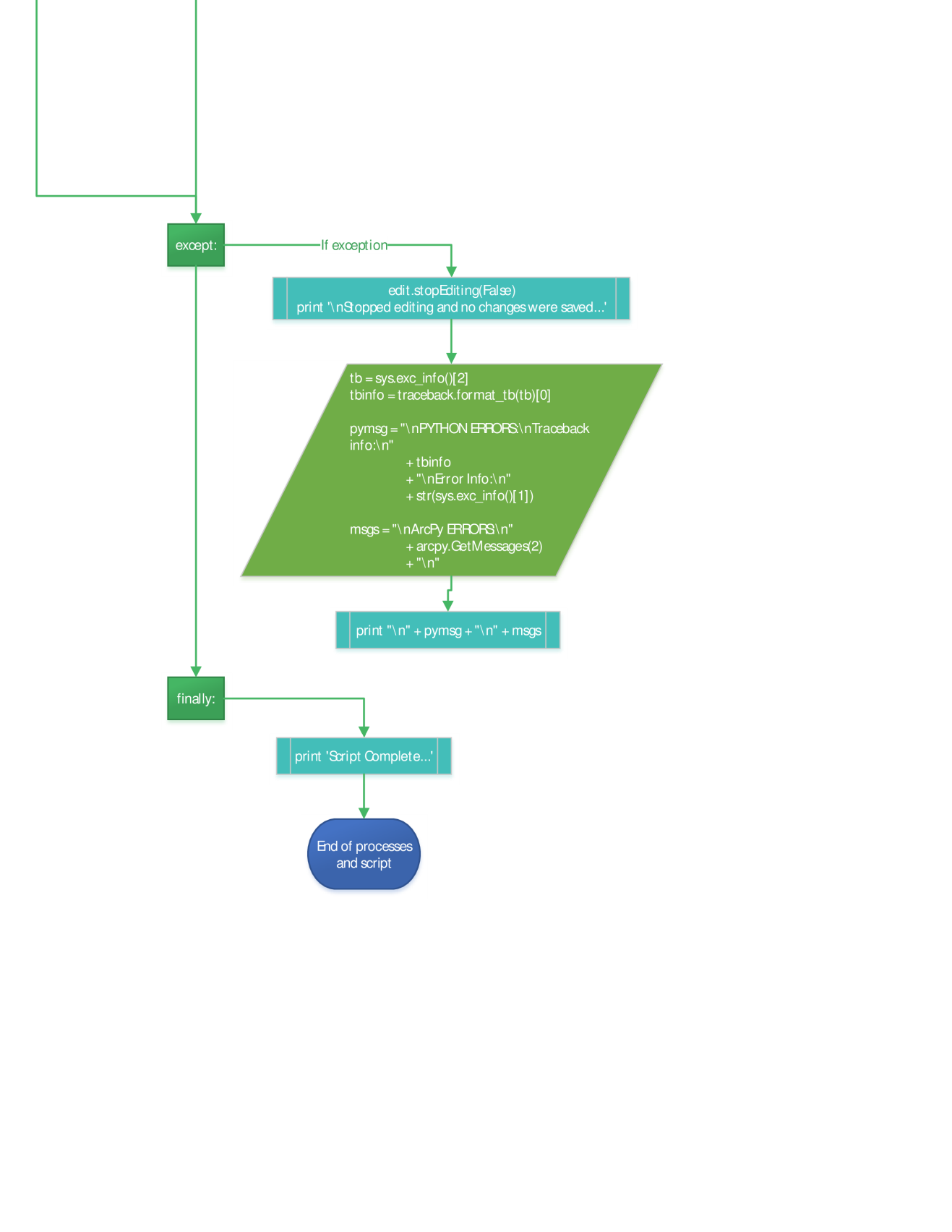
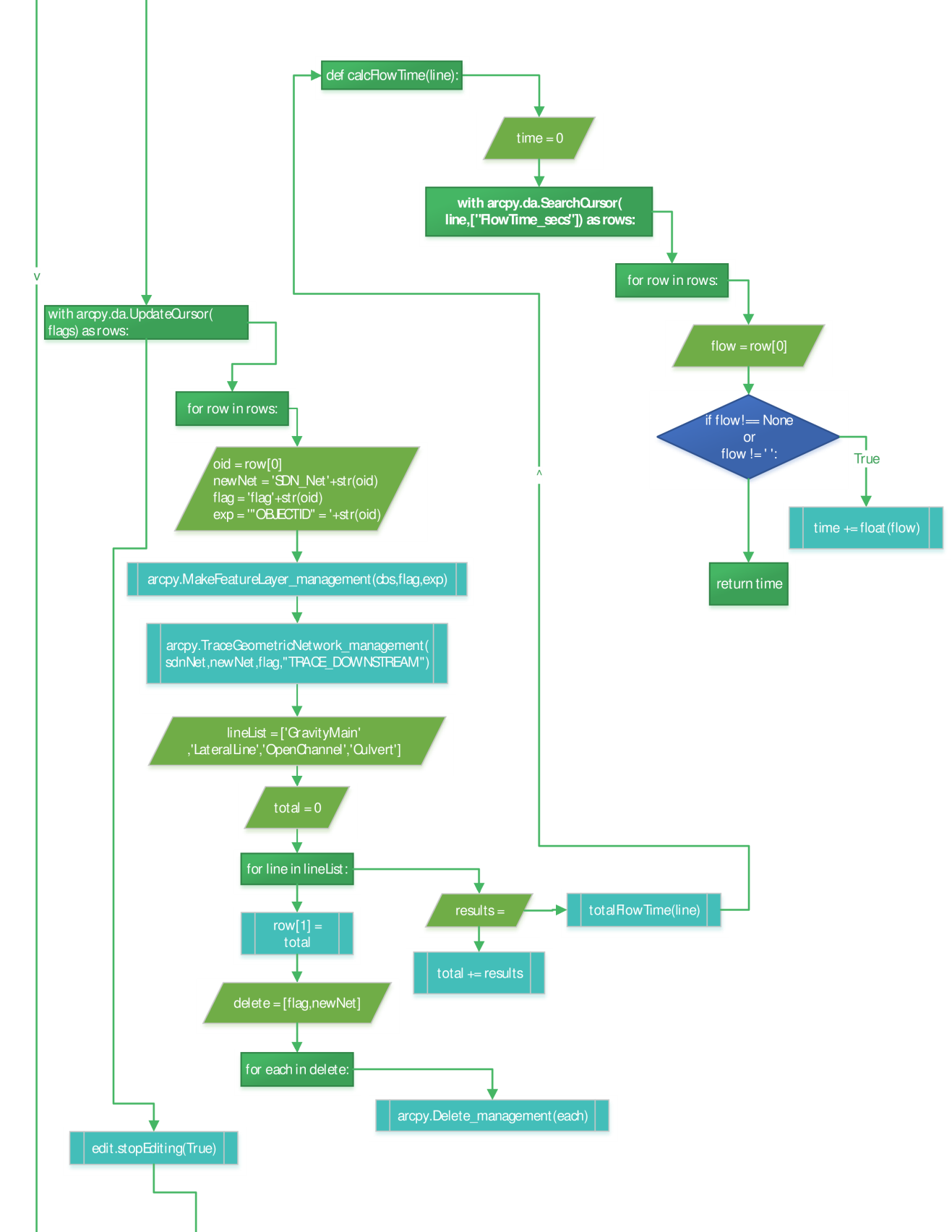
# Appendix A – Velocity.py



# C:\Users\ChadMarch\Desktop\velocity_Jan2014-1.pngC:\Users\ChadMarch\Desktop\velocity_Jan2014-2.png

# Appendix B

# C:\Users\ChadMarch\Desktop\Trace_Jan2014-0.png



# Appendix C

# Appendix D

# Appendix E

1. It’s important to note that there is no Federal standard of storm drain design. Each State of the Union sets their own standards for storm drain design in their Hydraulic Design Manuals. [↑](#footnote-ref-1)
2. United States Department of Agriculture TR-55 model: http://www.hydrocad.net/tr-55.htm [↑](#footnote-ref-2)
3. United States Army Corp of Engineers HEC model: http://www.hec.usace.army.mil/ [↑](#footnote-ref-3)
4. United States Geological Survey PRMS model: http://wwwbrr.cr.usgs.gov/projects/SW\_MoWS/PRMS.html [↑](#footnote-ref-4)
5. Manning’s Roughness “coef”ficients <http://www.engineeringtoolbox.com/mannings-roughness-d_799.html> [↑](#footnote-ref-5)